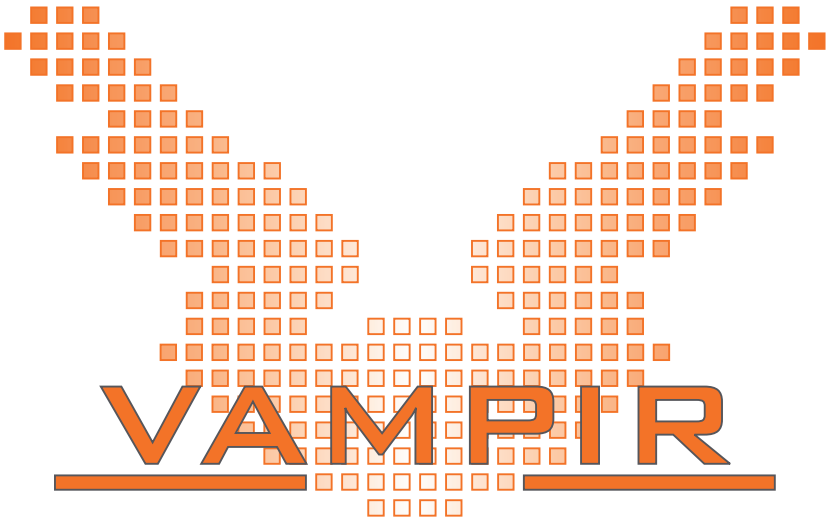
GWT forschung+innovation

VampirServer

User Manual



Copyright

© 2014 GWT-TUD GmbH

Blasewitzer Str. 43
01307 Dresden, Germany

http://gwtonline.de

Support / Feedback / Bug Reports

Please provide us feedback! We are very interested to hear what people like, dislike, or what features they are interested in.

If you experience problems or have suggestions about this application or manual, please contact service@vampir.eu.

When reporting a bug, please include as much detail as possible, in order to reproduce it. Please send the version number of your copy of *VampirServer* along with the bug report.

Please visit http://vampir.eu for updates and new versions.

service@vampir.eu
http://vampir.eu

Manual Version

VampirServer 8.4.0 / November 2014

Contents

1. Introduction **1**

2. Installation **3**

3. Vampir Server **7**
3.1. Using the Standard Control Interface 7
3.1.1. Obtaining a Command Overview (help) 7
3.1.2. Starting a New Server Instance (start) 8
3.1.3. Stopping an Existing Server Instance (stop) 10
3.1.4. Listing Server Related Information (list) 11
3.1.5. Configuring the Server (config) 11
3.1.6. Authentication 12
3.1.7. Command Line Reference 13
3.1.8. Environment Variables 14
3.2. Connecting Vampir to VampirServer 14
3.2.1. SSH Tunneling 15
3.3. Using the Back-End Control Interface 17
3.3.1. Manual Invocation 17
3.3.2. Environment Variables 18
3.4. Customizing Launch Scripts 19
3.4.1. Adding Batch Support (LSF) to the MPI Launch Script . . . 20
3.4.2. Adding Custom Launch Arguments 21

4. Vampir Proxy **23**
4.1. Prerequisites 23
4.2. Starting a Proxy Session 24
4.3. Connecting to a VampirServer Remote Station 25
4.4. Launching VampirServer via VampirProxy 26

A. Appendix **29**
A.1. Default MPI Launch Script 29
A.2. MPI Launch Script with LSF Support 32

1. Introduction

VampirServer is a software tool for analyzing the run-time behavior of parallel software programs. It visualizes the program execution by means of event traces, gathered by monitoring software like VampirTrace, Score-P, Scalasca, or TAU. The visualization takes place after the completion of the monitored program, by using data that has been captured during the program execution and stored in so-called trace files.

VampirServer is based on parallelized analysis algorithms. Data analysis and visualization are implemented as a client-server framework. The server component can be installed on a segment of a parallel production environment. The corresponding clients visualize the performance results graphically on remote desktop computers. Major advantages of this parallel and distributed approach are:

1. Performance data which tends to be bulky is kept at the location where it was created.
2. Parallel data processing significantly increases the scalability of the analysis process.
3. The applied performance analysis paradigm is easy to handle and works efficiently from arbitrary remote end-user platforms.
4. Very large trace files can be browsed and visualized interactively.

VampirServer translates a program's performance data into a variety of graphical representations providing developers with a good understanding of performance issues concerning their parallel and serial applications. VampirServer enables quick focusing on appropriate levels of detail which facilitates the detection and explanation of various performance bottlenecks such as load imbalances and communication deficiencies. This documentation is intended to be used both as a startup guide and as reference manual.

2. Installation

The platform specific executables for server and client, as well as their corresponding license files, are needed so as to run the VampirServer tool. Server and client are available in platform specific installation packages. Please, contact `sales@vampir.eu` for the purchase of software licenses. Once you obtained the installation files proceed with the following steps:

1. Start the VampirServer Installer by typing

   ```
   $ vampirserver-8.4.0-<platform>-setup.bin
   ```

   on your command line. You will be asked a few questions. Default answers to these questions are provided in brackets. Please confirm the default answers by pressing the *enter* key on your keyboard. Alternatively, you can enter the appropriate settings for your compupter system. Contact your system's administrator if you are not sure about the right answers.

   The VampirServer Installer copies the software into the directory <install-dir>. By default, <install-dir> is set to /opt/local/vampir-8.4.0 if installed with *root* permissions or $HOME/vampir-8.4.0 otherwise. After a successful installation, the following files reside in <install-dir>:

   ```
   INSTALL.txt
   bin/
       vampir-proxy
       vampirserver
       vampirserver-core
       vampirserver-shutdown
   doc/
       vampirserver-manual.pdf
       vampirserver-release-notes.txt
   etc/
       proxy/
       server/
   etc/proxy/
       agent
       agent-auto-start
       config
   etc/server/launcher/
   ```

```
        ap
        mpi
        smp
    etc/server/launcher/custom/
        llbg
        lsf
        mpi-intel
        pbs
        slurm
    lib/
        driver/
    lib/driver/
        config.h
        MpiModCore.c
        MpiModDef.h
        MpiModProto.h
```

In case of a configuration error, detailed error messages can be found in <install-dir>/vampirserver-config.log .

2. Adjust the system environment variable *PATH* as follows:

   ```
   $ export PATH=$PATH:<install-dir>/bin
   ```

   Make sure that this variable is set accordingly whenever you want to use VampirServer. This can be achieved by adding the above line to your login scripts or by creating an appropriate *software module* for the software. Contact your local system administrator for further details.

3. Copy the license file *vampir.license* that you received separately to <install-dir>/etc/ and make sure that it is readable (i. e. the right permissions bits are set) for all users and writable (!) for the user who is doing the setup procedure. See Section 3.3.2 for further details.

4. Start the VampirServer program with:

   ```
   $ vampirserver start smp
   ```

   This should result in a message similar to the following:

   ```
   Launching VampirServer...
   Estimating the number of processing elements
   ↪ (overwrite with -n option)...
   VampirServer 8.4.0
   Licensed to Fred Flintstone, Slate Rock and [...]
   Running 4 analysis processes...
   ↪ (abort with vampirserver stop 12991)
   VampirServer <12991> listens on: bedrock:30000
   ```

Certain Vampir license types require to be activated for individual computers. If this is the case for your license, VampirServer will print out the following message instead:

```
Launching VampirServer...

Dear customer, your license needs to be activated ...
Please, visit http://vampir.eu/activation ...

-----BEGIN REGISTRATION-----
LicenseId: 7592bbc4c9
Serial: 1146eca05c49d4b6b74a3daf5b046e92332930aa
-----END REGISTRATION-----

An activation file, named vampir.activation ...
Error: Could not start VampirServer.
```

Please visit `http://vampir.eu/activation` in order to obtain a license activation code file. Alternatively, send the printed registration information in an e-mail to `service@vampir.eu`. In both cases, an activation code file will be sent to you by e-mail. Copy the activation file *vampir.activation* to <install-dir>/etc/ and start over at this item.

5. Start the Vampir visualization program, which has to be installed independently from VampirServer. Establish a connection to VampirServer from within the visualization program by clicking on *Menu→File→Open Remote...*. Complete the input fields *Server* and *Port*. If both Vampir and VampirServer are executed on the same computer system you would typically have to enter *localhost* and a port number between 30000 and 30099. VampirServer's startup output

   ```
   VampirServer <12991> listens on: bedrock:30000
   ```

   assists you in finding the right connection parameters. It tells you exactly on which host VampirServer runs and to which port it listens.

   The progress and status of the connection setup is indicated in the Vampir performance visualizer. Once it is connected to a VampirServer instance, its usage is identical to the stand-alone Vampir program. Please consult the Vampir user manual for further reading.

6. Do further optional customization. The process of launching the VampirServer core program (vampirserver-core) is slightly platform dependent. The necessary parameters and environment variables are normally set automatically by the standard control interface (vampirserver), which makes

use of launch scripts located in <install-dir>/etc/server/launcher. If necessary, these scripts can be complemented with system specific settings like specific MPI or batch parameters. Example scripts are located in <install-dir>/etc/server/launcher/custom. Please note that this procedure addresses system administrators or experienced users.

3. Vampir Server

3.1. Using the Standard Control Interface

VampirServer is a parallel program that supports both distributed and shared memory computer systems. Unfortunately, the startup procedure of parallel programs on high performance computers is somewhat platform dependent. Therefore, VampirServer is shipped with a standard control interface that takes care of all platform dependent settings. The control interface is a command line program named *vampirserver*. This control interface is the preferred way to manage VampirServer analysis sessions. A default VampirServer instance can be started with the following command line sequence:

```
$ vampirserver start
```

which results in the following output:

```
Launching VampirServer...
VampirServer 8.4.0
Licensed to Fred Flintstone, Slate Rock and Gravel Inc.
Running 4 analysis processes...
↪ (abort with vampirserver stop 12991)
VampirServer <12991> listens on: bedrock:30000
```

The vampirserver command line program provides a generic interface to a set of administrative commands. Its general invocation syntax is as follows:

```
$ vampirserver [command] [arguments ...]
```

The following commands are supported: *help*, *start*, *stop*, *list*, *version*. We will now discuss the commands one by one. A short summary of all supported commands and arguments is given in Section 3.1.7.

3.1.1. Obtaining a Command Overview (help)

Issuing the following command on the command line:

```
$ vampirserver help
```

provides a brief overview of all commands and their arguments:

```
USAGE
    vampirserver [subcommand] [arguments ...] [-- [custom arguments ...]]

SUBCOMMANDS
    help, -h, --help             show this little help

    config, cf
        Interactively configures VampirServer for the given host system. MPI
        support can be enabled or disabled. The default launch script can be
        set.

        -s, --silent             use default answers for all questions

    list, ls  [servers | launchers]
        List server related information. Currently, this command lists all
        active servers or the available launch scripts (launchers). If no
        argument is provided, all active servers are listed.

    start, up  [-n] [-p] [-t] [LAUNCHER]
        Start a new server instance. LAUNCHER identifies the launch script to
        be used.

        -a, --auth=MODE          set authentication mode: crpw-rnd, none
        -n, --ntasks=NUMBER      set the number of analysis tasks
        -p, --port=NUMBER[:END]  set port (range) the server is going to listen on
        -t, --timeout=NUMBER     set the startup timeout to NUMBER seconds

    stop, ex  [SERVER_ID]
        Stop the given server or the most recent server if no SERVER_ID is
        provided. The server ID is printed during startup. Alternatively, use
        the list command to print a list of available servers.

    version, -v, --version   show VampirServer's revision
```

3.1.2. Starting a New Server Instance (start)

Issuing the following command on the command line:

```
$ vampirserver start
```

starts a new instance of VampirServer as a background process. Upon startup, VampirServer will generate the following output:

```
Launching VampirServer...
VampirServer 8.4.0
Licensed to Fred Flintstone, Slate Rock and Gravel Inc.
Running 4 analysis processes...
↪ (abort with vampirserver stop 12991)
VampirServer <12991> listens on: bedrock:30000
```

The last output line provides the server ID (12991), the host address (bedrock), and the socket port (30000) where the server awaits connection requests from the Vampir performance visualizer.

By default, VampirServer will start with a concurrency level of four analysis tasks. For analysis sessions with large data volumes this concurrency level can be increased with the option *-n*:

```
$ vampirserver start -n 32
```

which starts a new instance of VampirServer with 32 parallel worker tasks. VampirServer will generate the following output in return:

```
Launching VampirServer...
VampirServer 8.4.0
Licensed to Fred Flintstone, Slate Rock and Gravel Inc.
Running 32 analysis processes...
↪ (abort with vampirserver stop 13227)
VampirServer <13227> listens on: bedrock:30000
```

Please note that it is recommended to have at least as many cores in your computer system as the specified concurrency level. Otherwise, a performance degradation will be observed.

During its lifetime VampirServer listens on a specific socket port for incoming connection requests from the Vampir performance visualizer. At startup VampirServer picks a free port in the range of 30000-30099. Sometimes it is desirable to use one specific port or a different port range. The port selection behavior can be modified with the option *-p*. Issuing the following command on the command line:

```
$ vampirserver start -p 47011
```

starts a new instance of VampirServer that listens to port number 47011. VampirServer generates the following output in return:

```
Launching VampirServer...
VampirServer 8.4.0
Licensed to Fred Flintstone, Slate Rock and Gravel Inc.
Running 32 analysis processes...
↪ (abort with vampirserver stop 17653)
VampirServer <17653> listens on: bedrock:47011
```

If port 47011 is already in use by another application, the invocation of VampirServer fails with the following error message:

```
Error: Failed to listen to network port 47011
```

Depending on the host platform, the startup of VampirServer involves the allocation of system resources through the installed batch system. The initialization and provision of system resources can induce long delays if they are temporarily unavailable. VampirServer's startup process timeouts if system resources are unavailable for a certain period of time. The respective waiting time in seconds can be set with the option *-t*. The startup command

```
$ vampirserver start -t 60
```

terminates after 60 seconds if the allocation of system resources could not be achieved. It generates the following output:

```
Launching VampirServer...
Error: Could not start VampirServer.
```

The invocation an initialization of VampirServer depends on host specific characteristics like the MPI system, the batch system, or the memory and processor architecture. The required checks and adjustments are hidden in fully transparent launch scripts so that host specific customization is reduced to a minimum. VampirServer automatically uses a default launch script, which is selected during the software installation process. Sometimes, additional launch scripts are beneficial to support multiple system setups. If required, the user can manually select a specific launch script by adding the launch script's name to the invocation command.

```
$ vampirserver start smp
```

launches VampirServer in thread mode, i. e. MPI parallelization is disabled and shared memory parallelization with threads is used instead. Currently, three default launch scripts are shipped with VampirServer:

- **ap:** is a pure MPI startup sequence based on aprun for Cray systems.
- **mpi:** provides a pure MPI startup sequence without batch job creation.
- **smp:** is an alternative startup sequence for a threaded shared memory execution without MPI parallelization and without batch job creation.

Launch scripts can customized by system administrators and users. See Section 3.4 for further details. By default, VampirServer searches for script files in <install-dir>/etc/server/launcher. Alternatively, an absolute file path with a leading / to an arbitrary location in the file system can be specified as launch script argument.

The server program is executed as a background process. It runs until it is terminated either manually (see Section 3.1.3 below) or automatically by the host computer's batch system.

3.1.3. Stopping an Existing Server Instance (stop)

The standard control interface keeps track of all VampirServer launches. Upon startup, a specific server ID is assigned to each server instance and printed as follows:

```
VampirServer <server-id> listens on: <host>:<port>
```

A given active server instance can be terminated by issuing a stop command with the respective server ID. Issuing the following command on the command line will stop server instance 9991:

```
$ vampirserver stop 9991
Shutting down VampirServer <9991>...
VampirServer <9991> is down.
```

Please note that the server ID is an optional argument. If it is omitted, the standard control interface will terminate the most recently launched server instance. Repeated invocation of `vampirserver stop` will terminate server instances one by one until no active servers are left.

3.1.4. Listing Server Related Information (list)

The standard control interface of VampirServer can list related status and setup information of program instances. Currently, two list types are supported: the *servers* list includes all VampirServer instances that have been started by the user. Alternatively, the *launchers* list provides the names of all pre-defined launch modules.

The following command on the command line will list all known server instances:

```
$ vampirserver list servers
24947 mars:30055 [4x, mpi]
24948 neptun:30056 [16x, smp]
24949 mars:30057 [8x, ap]
```

The output lines have the following format:

<server-id> <host-name>:<port-number> [<ntasks>, <launcher>].

Each line starts with a *server-id*, which is a unique number for every server instance. It is followed by the network location of VampirServer's master service process. The information consists of the *host* name and the *port* number separated by a colon. The Vampir performance visualizer requires this information during a connection setup. A line ends with brackets enclosing the degree of parallelism (*ntasks*) and the name of the *launcher* startup module for a particular server instance.

3.1.5. Configuring the Server (config)

The VampirServer program needs to be configured for the given host system prior to being used for the first time. Normally, this configuration is done during

the software installation (see Section 2). It is however possible to re-configure VampirServer at any time[1] by typing:

```
$ vampirserver config
```

on the command line. Re-configuration can become necessary if default startup parameters need to be changed or if the system's message passing library (MPI) has been updated or replaced. During the configuration the following questions will be asked:

- Would you like to enable MPI support in VampirServer? [y]
- MPI Compiler used for VampirServer customization [/usr/bin/mpicc]:
- Compiler flags for shared object creation [-shared -fPIC]:
- Default VampirServer launch configuration (ap, mpi, smp) [mpi]:

Default answers to these questions are provided in brackets. The default answers can be confirmed by simply pressing the 'enter' key on the keyboard. Alternatively, the appropriate system settings can be entered.

For automatic, non-interactive configuration, the command line option *--silent* can be set. The resulting configuration assumes default values for all parameters.

3.1.6. Authentication

VampirServer has encrypted password support for client (Vampir) authentication[2]. Authentication can be activated or deactivated with the *-a* command line option. Issuing the following command on the command line:

```
$ vampirserver start -a crpw-rnd
```

starts a new instance of VampirServer that requires password authentication during the connection setup with Vampir. VampirServer will generate the following output:

```
Launching VampirServer...
VampirServer 8.4.0
Licensed to Fred Flintstone, Slate Rock and Gravel Inc.
Running 4 analysis processes...
↪ (abort with vampirserver stop 12991)
User: fred
Password: jfvUzFHJaW+p
VampirServer <12991> listens on: bedrock:30000
```

[1] System administrator rights might be necessary
[2] Please note that the data transfer protocol itself is not encrypted.

User is the login name of the user who started VampirServer. *Password* is a random password that needs to be entered on the client side in order to connect to the given instance of VampirServer.

Authentication can be disabled by starting VampirServer with the following command line sequence:

```
$ vampirserver start -a none
```

3.1.7. Command Line Reference

The standard control interface of VampirServer is a powerful front-end for managing service instances. It hides platform dependent setup steps from the software user. Table 3.1 gives a brief overview of the commands that are understood by the vampirserver command line tool.

Commands	**Arguments and Description**	
config, cf	[-s]	Configure server settings
	-s, --silent	Use defaults for all questions
help	Show a brief command overview	
start, up	[-a] [-n] [-p] [-t] [SCRIPT] [--]	Start a new server instance
	-a, --auth MODE	Set the authentication mode
	-n, --ntasks NUMBER	Set the number of analysis tasks
	-p, --port NUMBER[:END]	Set port (range) the server is going to listen on
	-t, --timeout NUMBER	Set the startup timeout to NUMBER seconds
	SCRIPT	Name of the launch script to be used
	--	Separator for custom launch script parameters
stop, ex	[SERVER_ID]	Stop a given server instance
	SERVER_ID	Sever specific ID assigned during startup
list, ls	[servers | launchers]	List server related information
	servers	List all launched servers
	launchers	List all available launch modules
version	Show program version	

Table 3.1.: Commands and arguments of the VampirServer control interface

3.1.8. Environment Variables

The VampirServer standard control interface evaluates the following environment variables. These variables overwrite the built-in defaults. Use them with care.

VAMPIRSERVER_HOST

If VampirServer is supposed to run on multiple hosts that share home directories this variable is applicable. Setting it to the respective host name guarantees that Vampir instances on different hosts are managed in separate lists.

VAMPIRSERVER_LAUNCHER

Sets the dafault launch script that is used when no script name is given during startup. The following command sequence

```
$ export VAMPIRSERVER_LAUNCHER="ap"

$ vampirserver start
```

has the same effect as

```
$ vampirserver start ap
```

This variable can be handy on systems that support custom software modules.

3.2. Connecting Vampir to VampirServer

The VampirServer program has to be used in combination with the Vampir performance data browser, which can connect to multiple instances of VampirServer. Once a connection has been established between Vampir and VampirServer, trace files containing performance data can be read and analyzed.

A new remote session can be created by clicking on *Open Remote...* in the *File* menu. As a result, an input dialog appears as depicted in Figure 3.1. On the right hand side, the server's host name, network port, and authentication mode can be specified. The default parameters are *localhost*, *port 30000*, and *no authentication (None)*. Clicking on the *Connect* button starts the connection setup. Once the connection is established, a file dialog is opened, which allows to browse the remote files on VampirServer's host system as depicted in Figure 3.2. The trace files to be loaded have to be compliant with the *OTF* or the *Epilog* trace file format. A compliant trace file can be loaded by selecting the respective file name and clicking on the *Open* button.

After loading has been completed, Vampir will depict the *Master Timeline*, the *Function Summary*, and the respective *Function Legend*. Please consult the Vampir manual for further information. Recently viewed trace files can be reopened via *Menu*→*File*→*Open Recent*.

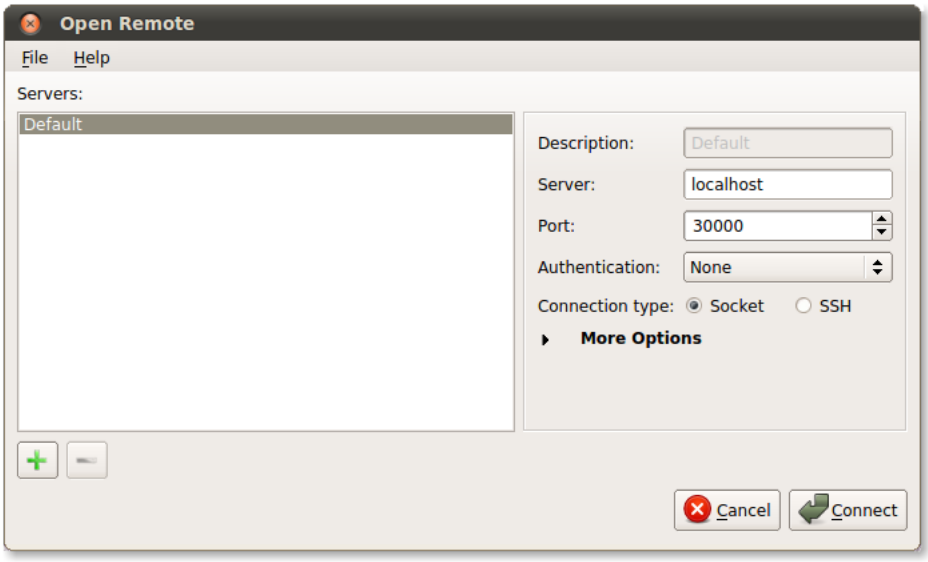


Figure 3.1.: Connect to Server Dialog

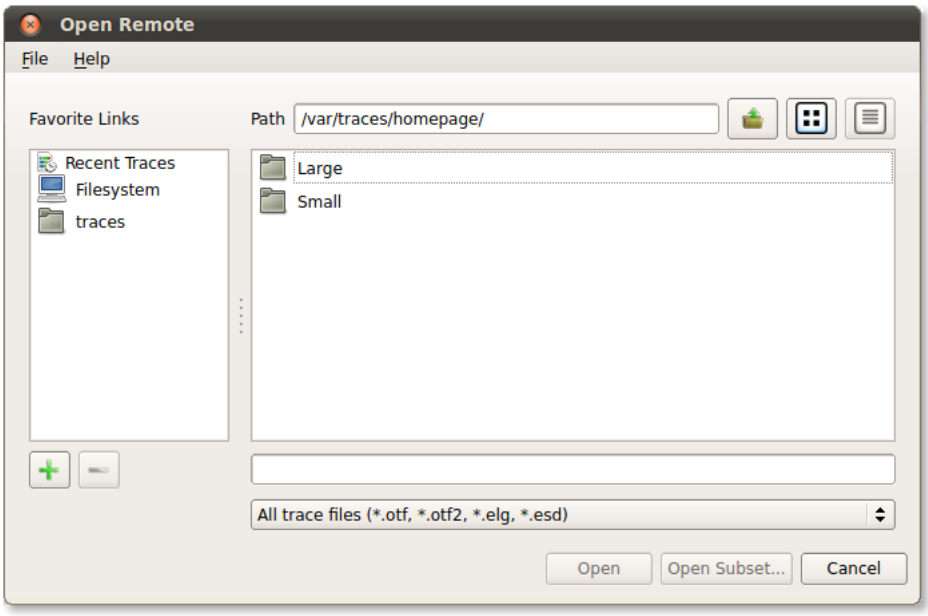


Figure 3.2.: Open Remote Trace File Dialog

3.2.1. SSH Tunneling

Network firewalls often prohibit to directly connect to an active VampirServer instance on a given port like 30000. In such a situation *ssh tunneling* can be used to set up a secure network tunnel from a local computer running the Vampir browser to the remote computer where VampirServer is active. The following command line sequence sets up a network tunnel from *local port* on your local

computer to the *remote port* of the given *remote computer*.

```
$ ssh -L <local-port>:<remote-node>:<remote-port>
↪ [<user>@]<remote-computer>
```

Remote port needs to be set to the port number printed at the startup of VampirServer:

```
VampirServer <server-id> listens on:
↪ <remote-node>:<remote-port>
```

The parameter *user* is optional and specifies your login name on the remote computer. Please consult the *SSH* manual for further information about network tunnels. Once the network tunnel is set up, the Vampir performance browser needs to be connected locally to *localhost* on *local port*.

The following example starts a VampirServer instance on the remote computer *bedrock.eu* and sets up a tunnel from an aribtrary local computer. First, VampirServer is started with:

```
bedrock$ vampirserver start
Launching VampirServer...
VampirServer 8.4.0
Licensed to Fred Flintstone, Slate Rock and Gravel Inc.
Running 4 analysis processes...
↪ (abort with vampirserver stop 13227)
VampirServer <13227> listens on: node42:30000
```

The following command line sequence on the local computer sets up an SSH tunnel to *node42* on the remote computer *bedrock.eu*:

```
local$ ssh -L 30001:node42:30000 bedrock.eu
```

Finally, Vampir (on the local computer) is connected to the remote server by means of the *Open Remote* dialog which appears when klicking on *File→Open Remote...* in Vampir's main menu. Prior to clicking on the *Connect* button, *localhost* and *30001* needs to be entered in the respective input fields *Server* and *Port*.

Unfortunately, the setup process of a tunneled Vampir/VampirServer session is somewhat cumbersome. Therefore, Vampir 7.5 introduced an alternative remote connection type (SSH) that renders this user driven setup process obsolete. It involves a new component called *VampirProxy*, which automatically takes care of the above steps. See Section 4 for further information.

3.3. Using the Back-End Control Interface

VampirServer's processing core is a parallel distributed program whose invocation slightly depends on the host platform. These dependencies are handled automatically by the *standard control interface* described in Section 3.1. Therefore, it is recommended to use VampirServer's standard control interface by default. However, it is possible to directly start the VampirServer processing core from the command line. Please note that this is only recommended to experienced users or system administrators. This section describes the server core's invocation syntax and options.

3.3.1. Manual Invocation

The server core currently supports two modes of parallel operation: MPI mode and thread mode. For the former, MPI has to be installed and configured properly before starting the server. The exact command line is MPI implementation-dependent. If *LAM-MPI* is used, *lamboot* must be executed on the command line prior to using mpirun. If MPICH is used, a machine file might be needed (see MPICH user manual). Other MPI implementations might require different steps. Please consult the MPI documentation of your computer system.

A server core instance can be started manually with the following command line sequence:

```
$ mpirun -np <number of processes>
↪ <install-dir>/bin/vampirserver-core
```

or

```
$ mpiexec -np <number of processes>
↪ <install_dir>/bin/vampirserver-core
```

Please note that the number of MPI processes must at least be two. Its recommended upper boundary is the number of processes and threads in the trace files to be processed plus one. The "plus one" results from the fact that VampirServer uses a master/worker data processing model with one master and n worker tasks.

The thread mode of the server core needs to be invoked **without** *mpirun* or a similar prefix. Simply type:

```
$ export VAMPIRSERVER_MODE="thread-mode"
$ vampirserver-core
```

The server core will automatically detect the optimal number of threads for your system. Alternatively, the number of threads can be set manually with the command line option *-n*.

-a	--auth=MODE	Authentication mode: *crpw-rnd* or *none*
-c	--chost=NAME	Cluster node that is going to listen for requests
-h	--help	Show this help
-n	--nthreads	Number of analysis threads (1–16) if VAMPIRSERVER_MODE is set to *thread-mode*
-p	--port=NUMBER:[END]	Port range, the server is going to listen for requests
-v	--version	Show program version

Table 3.2.: Command line options of the vampirserver-core program

A summary of all supported command line options of the server core is given in Table 3.2:

The server program will run until it is terminated either manually with the key sequence *Ctrl-C* on the command line or automatically by the host's batch system. Alternatively, a small utility program is provided that triggers an internal shutdown of the server program. On systems without automatic MPI cleanup this utility can help to ensure that no orphaned processes remain on the system. Type

```
$ vampirserver-shutdown -p <host>:<port>
```

on the command line to trigger a server core shutdown. The server core will terminate with the following output:

```
Server shutdown triggered by client.
```

3.3.2. Environment Variables

The VampirServer core evaluates the following environment variables for configuration purposes. These variables overwrite the built-in defaults. Use them with care. Please note that some MPI implementations require special command line options, e.g. -x VAMPIRSERVER_DRIVER, to correctly propagate environment variables among their processes. Consult the documentation of your MPI installation for further details.

VAMPIRSERVER_MODE

The server core currently supports two modes of parallel operation: MPI mode and thread mode. Setting the environment variable VAMPIRSERVER_MODE to *thread-mode* with

```
$ export VAMPIRSERVER_MODE="thread-mode"
```

enables the built-in thread parallelization of the vampir core. MPI parallelization is enabled by setting VAMPIRSERVER_MODE to *mpi-mode* with

```
$ export VAMPIRSERVER_MODE="mpi-mode"
```

The server core will start in MPI mode if the variable VAMPIRSERVER_MODE is not defined or set to an unknown string.

VAMPIRSERVER_DRIVER

In MPI mode, the server core requires an MPI specific driver module named *vampirserver-driver.so* that is normally setup during installation. Speaking technically, this driver module is a dynamically linked shared object library. By default, the module is located in <install-dir>/lib/vampirserver-driver.so. The environment variable VAMPIRSERVER_DRIVER can be used to overwrite this default location.

Use the following command line to set an alternative file name and path:

```
$ export VAMPIRSERVER_DRIVER="my-path/my-driver.so"
```

This option can be used to support multiple MPI implementations on the same host. Please note that such a setup requires expert knowledge about the host system.

VAMPIRSERVER_LICENSE, VAMPIR_LICENSE

The server core requires a valid Vampir Professional license. After a standard installation, the license file is located in <install-dir>/etc/vampir.license. An alternative file name and path can be set with either VAMPIRSERVER_LICENSE or VAMPIR_LICENSE with the following command line:

```
$ export VAMPIRSERVER_LICENSE="my-path/my-vampir.license"
```

The variable VAMPIRSERVER_LICENSE has precedence over VAMPIR_LIC ENSE if both variables are defined. Please note that both variables also influence the respective license activation file location, which normally is *<install-dir>/etc/vampir.activation*. The alternative license file name above for example results in the following activation file location: *my-path/my-vampir.activation*.

3.4. Customizing Launch Scripts

The invocation an initialization of VampirServer depends on host specific characteristics like the MPI system, the batch system, or the memory and processor architecture. The required checks and adjustments are hidden in fully transparent

launch scripts so that host specific customization is reduced to a minimum. VampirServer automatically uses a default launch script, which has been selected during the software installation process. Sometimes, additional launch scripts are beneficial to support multiple system setups. If required, the user can manually select a specific launch script by adding the launch script's name to the invocation command as described on page 10. Additionally, custom launch arguments like a *batch job reservation number* can be forwarded to the launch scripts. Example launch scripts are located in <install-dir>/etc/server/launcher/custom.

3.4.1. Adding Batch Support (LSF) to the MPI Launch Script

This section describes the customization of a launch script step by step. Based on the default MPI launch script a new script with support for the LSF batch system is created. The default MPI launch script is located at *<install-dir>/etc/server/launcher/mpi*. Its full listing is available at A.1. The modified launch script with LSF support is listed in A.2. The following steps need to be done to add LSF support to the default MPI launch script:

1. Batch jobs usually have to wait some time before they are executed. Therefore the timeout is increased from five seconds to 300 seconds in line 17.

2. Starting VampirServer as a batch job is LSF specific. Lines 21 to 25 have been extended to use LSF for job submission.

Listing 3.1: Startup Snippet with Pure MPI Support

```
16  # Startup timeout in seconds.
17  TIMEOUT=5
18
19  # Launch vampir server process.
20  launch_vs() {
21      ${MPIRUN} ${PREFIX}/bin/vampirserver-core ${opt} >${tmpfile} 2>&1 &
22      disown
23
24    # Return process ID or job ID as shutdown reference.
25      OUT_CUSTOM=$!
26  }
```

Listing 3.2: Startup Snippet with LSF Support

```
16  # Startup timeout in seconds.
17  TIMEOUT=300
18
19  # Launch vampir server process.
20  launch_vs() {
21      local submission; echo "Submitting LSF batch job (this might take a while)...
            "
22      submission=`bsub -n $((${IN_NTASKS} + 1)) -o ${tmpfile} ${MPIRUN} ${PREFIX}/
            bin/vampirserver-core ${opt}`
23
24    # Return process ID or job ID as shutdown reference.
25      OUT_CUSTOM=`echo "${submission}" | grep "is submitted" | sed "s/^Job <//;s/>
            is submitted.*//"`
26  }
```

3. Stopping a VampirServer instance is LSF specific likewise. Lines 32 to 35 have been altered to properly terminate the respective LSF job.

Listing 3.3: Snippet with MPI Shutdown

```
29 # Kill vampir server process.
30 kill_vs() {
31   # Kill server process/job.
32     kill -9 ${IN_CUSTOM} 2>/dev/null
33
34   # Wait for server to terminate.
35     while [ $(kill -0 "${IN_CUSTOM}" 2>/dev/null) ]; do
36         sleep 1
37     done
38 }
```

Listing 3.4: Snippet with LSF Shutdown

```
29 # Kill vampir server process.
30 kill_vs() {
31   # Kill server process/job.
32     bkill >/dev/null ${IN_CUSTOM}
33
34   # Wait for server to terminate.
35     while false; do # No waiting needed! Already done by bkill.
36         sleep 1
37     done
38 }
```

The resulting LSF launch script needs to be saved to <install-dir>/etc/server/launcher/lsf. It can be tested by adding the launch script's file name to VampirServer's start command.

```
$ vampirserver start lsf
```

User-defined launch scripts (not located in <install-dir>/etc/server/launcher) are supported as well. In this case, an absolute file path with a leading slash has to be added to the start command.

```
$ vampirserver start /tmp/custom-script
```

3.4.2. Adding Custom Launch Arguments

VampirServer's launch scripts take care of the parallel startup process for different system types and configurations. For portability reasons, they are generic and use default system parameters, for example the default batch system queue or the default MPI run-time system. Some systems require users to provide user specific information for job submission like certain batch queues, reservation codes, or accounting information. VampirServer's standard control interface can forward this kind of user specific information to the actual launch script in the form of custom arguments. Custom arguments need to be separated from standard arguments by means of two dashes (--). The following command line sequence starts VampirServer and hands over the custom argument *-a* providing the value 4711.

```
$ vampirserver start -n 4 mpi -- -a 4711
```

During startup, the server will printout the following:

```
Launching VampirServer...
Custom argument A = 4711
VampirServer 8.4.0
Licensed to Fred Flintstone, Slate Rock and Gravel Inc.
Running 4 analysis processes...
↪ (abort with vampirserver stop 12991)
VampirServer <12991> listens on: bedrock:30000
```

All example launch scripts are pre-equipped with reading support for custom arguments. Its custom application needs to be added manually to a launch script according to the individual demands. Please note that this is an advanced task which addresses system administrators or experienced users. The relevant parts in the launch scripts are:

Listing 3.5: Custom Argument Parser in MPI Launch Script

```
50 # Parse non-Vampir command line arguments for customization.
51 IN_CUSTOM_ARG_A=""
52 IN_CUSTOM_ARG_B=""
53 parse_custom_args()
54 {
55     temp=`getopt a: "$@"`
56     if [ $? != 0 ] ; then echo "Terminating..." >&2 ; exit 1 ; fi
57     eval set -- "$temp"
58     while true ; do
59         case "$1" in
60             -a) IN_CUSTOM_ARG_A="${2}"; shift 2
61                 ;;
62             --) shift; break
63                 ;;
64             *)  echo "Error: Internal!" >&2; exit 1
65                 ;;
66         esac
67     done
68     if [ -n "$1" ]; then
69         IN_CUSTOM_ARG_B="$1"
70     fi
71 }
```

and

Listing 3.6: Custom Argument Application in MPI Launch Script

```
137    # Example custom arguments.
138     parse_custom_args ${IN_CUSTOM}
139     if [ -n "${IN_CUSTOM_ARG_A}" ]; then
140         echo "Custom argument A = ${IN_CUSTOM_ARG_A}"
141     fi
142     if [ -n "${IN_CUSTOM_ARG_B}" ]; then
143         echo "Custom argument B = ${IN_CUSTOM_ARG_B}"
144     fi
```

Please add your customizations accordingly.

4. Vampir Proxy

VampirProxy is a command line tool that supports the user in establishing a connection between the Vampir performance visualizer running on a local desktop computer and a remote instance of the VampirServer performance processor. It fully automates the setup of so-called *communication tunnels*. High performance computing resources usually require such communication tunnels for remote data visualization. VampirProxy can automatically launch and connect to remote VampirServer instances.

4.1. Prerequisites

First of all, make sure that the *vampir-proxy* program is installed properly on your local system *and* the remote computer system. Also make sure that vampir-proxy exists in the search path on the remote computer system. You can verify this by typing:

```
$ ssh <remote-address> vampir-proxy
```

on the command line of your local computer system. The following output will be printed to your console if the invocation was successful:

```
USAGE
    vampir-proxy [OPTIONS] [[user@]hostname]

OPTIONS
    Local options, evaluated when hostname is set:
    -C, --config-local=FILE             local configuration file
    -P, --proxy-port=NUMBER             local proxy listen port (default: 30000)
    -E, --rsh=COMMAND                   remote shell invocation (default: ssh)
                                        (default: use remote configuration file)

    Options which are forwarded/executed on the remote side:
    -c, --config=FILE                   configuration file on the remote side
                                        (default: $HOME/.vampir/proxy/config,
                                        <install-dir>/etc/proxy/config)
    -s, --config-section=NAME           section in the configuration file
                                        (default: DEFAULT)
    -h, --help                          show this help
    -l, --list                          list active servers on the remote side
        --proxy-exec=PROGRAM            proxy file path on the remote side
                                        (default: $PATH/vampir-proxy)
    -a, --server-host=ADDRESS           server address (default: localhost)
    -p, --server-port=NUMBER            server listen port (default: 30000)
        --server-script=SCRIPT          script that starts vampir server
                                        (default: $HOME/.vampir/proxy/agent,
                                        <install_dir>/etc/proxy/agent)
        --server-startup=MODE           start server on remote host
```

```
                                    [no][single][multi] (default: no)
  -n, --server-tasks=NUMBER         number of worker tasks (excluding boss)
                                    (default: 4)
  -v, --verbose                     increase verbosity
```

If the command vampir-proxy has not been found on the remote side, the following output will be printed to your console:

```
bash: vampir-proxy: command not found
```

Make sure that the install path of vampir-proxy is permanently added to your default search path (see the shell documentation about the PATH environment variable on your remote computer).

Finally, check that VampirServer is installed properly on your (remote) computer system by typing:

```
$ vampirserver start
```

which should result in an output similar to:

```
Launching VampirServer...
VampirServer 8.4.0
Licensed to Fred Flintstone, Slate Rock and Gravel Inc.
Running 4 analysis processes...
↪ (abort with vampirserver stop 12991)
VampirServer <12991> listens on: bedrock:30000
```

See Section 2 for the installation of VampirServer.

4.2. Starting a Proxy Session

VampirProxy needs to be started on the computer system that runs the Vampir performance visualizer. In the following section, this computer system will be referred to as *local system*. The *remote system* is the machine where VampirServer will be running. The following input on the command line interface will start a proxy instance on the local system:

```
$ vampir-proxy [<user>@]<host>
```

The proxy command on the local system will automatically establish a connection to the remote system *host*. A successful connection setup is quoted with the following message:

```
Waiting for confirmation from the remote side.
This may take some time.
Listening on localhost:30000
Type "quit" and press "Enter" to exit vampir-proxy.
```

On the local system you can now connect to the remote VampirServer instance by connecting to localhost:30000 (see Section 3.2 for further details). An error is indicated as follows:

```
VAMPIR-PROXY: Error: Could not find the vampir-proxy
↪ executable on the remote host.
```

On the local system, VampirProxy listens on network port 30000 for incoming connect request from the Vampir performance visualizer. The port number can be changed with the command line option *-P <port>*.

In order to connect to a remote system, VampirProxy uses the remote-shell program *ssh* as the transport. It is possible to set a different remote-shell program and additional command line arguments with the option *-E "<command>"*. The proxy invocation

```
$ vampir-proxy -P 30003 -E "ssh -c blowfish" bedrock.eu
```

for example, uses the remote shell program *ssh* with *blowfish* encryption for transport and offers its service on the local network port 30003. The connection is established to a remote system named *bedrock.eu*.

4.3. Connecting to a VampirServer Remote Station

When connected to a remote system, VampirProxy tries to forward all requests to the remote VampirServer instance listening on port 30000. For this basic example, we assume that VampirServer was manually started beforehand (see Section 3.1.2). It is possible to alter this default behavior with the command line options *-p <port>* and *-a <host>*, which can be used to contact an alternative server *on* or *reachable from* the remote system.

Entering the following command line sequence on the local system sets up a proxy session between port 300xx (default) on the local system and port 25000 on the remote system called *bedrock.eu*.

```
$ vampir-proxy -p 25000 bedrock.eu
```

This example assumes that VampirServer has been started manually beforehand on *bedrock.eu* and that it has been configured to listen on port 25000 (see Section 3.1.2).

If required, the local port of VampirProxy can be changed with the *-P* (capital!) option. Entering

```
$ vampir-proxy -p 25000 -P 25001 bedrock.eu
```

on the local system starts a proxy session that connects to a VampirServer instance on *bedrock.eu* port 25000 and forwards its service to the local system on port 25001.

Large computer systems often consist of login and compute nodes. The latter are typically protected and hidden behind a firewall. VampirProxy can also connect to VampirServer instances behind a firewall. The following example assumes that a VampirServer instance is active on the compute node *node42* of the remote system *bedrock.eu*. The following command line sequence starts a proxy session to VampirServer on node42 (port 30000) via the login node bedrock.eu.

```
$ vampir-proxy -a node42 bedrock.eu
```

On some remote systems, the executable *vampir-proxy* might not be in the default search path when invoked as SSH remote command. VampirProxy will consequently fail with the following error message:

```
VAMPIR-PROXY: Error: Could not find the vampir-proxy
↪ executable on the remote host.
```

The problem usually can be solved by extending the default search path on the remote system. If this is not possible or does not solve the problem for some unknown reason, it is possible to specify the absolute remote path of vampir-proxy as follows:

```
$ vampir-proxy --proxy-exec /home/barnie/vampir-proxy
↪ bedrock.eu
```

4.4. Launching VampirServer via VampirProxy

VampirProxy facilitates the connection setup between the Vampir performance browser and VampirServer. The previous sections assume that VampirServer is started manually by the user. VampirProxy can also automatically launch program instances of VampirServer. The command line options *--server-startup* and *--server-script* control the launch behavior, which is turned off by default. Prior to using this functionality, make sure that VampirServer is reachable and configured on the remote system by typing:

```
$ ssh <remote-address> vampirserver start
```

on the command line of the local computer system. The resulting output should look like:

```
Launching VampirServer...
Estimating the number of processing elements
↪ (overwrite with -n option)...
VampirServer 8.4.0
Licensed to Fred Flintstone, Slate Rock and Gravel Inc.
Running 4 analysis processes...
↪ (abort with vampirserver stop 12991)
VampirServer <12991> listens on: bedrock:30000
```

The following command line sequence on the local computer system creates a proxy connection to the remote computer system *bedrock.eu* and automatically starts a new instance of the VampirServer program.

```
$ vampir-proxy --server-startup=multi bedrock.eu
```

A successful setup is prompted with the following message:

```
Waiting for confirmation from the remote side.
This may take some time.
Listening on localhost:30000
Type "quit" and press "Enter" to exit vampir-proxy.
```

The command line option *--server-startup=multi* implies that each proxy session starts its a own instance of the VampirServer program. This is not desirable in all situations. Alternatively, *--server-startup=single* starts only one instance of the VampirServer program that is shared among other sessions started with this option. Finally, *--server-startup=no* prohibits automatic startup of VampirServer, even if it has been made the default configuration during software installation.

By default, VampirServer is started with 1 to 16 parallel tasks depending on the available CPU resources. It is possible to set the number of tasks manually with the command line option *--server-tasks*. Entering the command line sequence

```
$ vampir-proxy --server-tasks=32 --server-startup=multi
↪ bedrock.eu
```

on the local system sets up a proxy session with *bedrock.eu* and starts a VampirServer instance with 32 parallel analysis tasks on *bedrock.eu*. Please note that this option requires *--server-startup=multi* to take effect. When combined with *--server-startup=single* this option will only influence the initial startup of the shared VampirServer instance. With *--server-startup=no* it has no effect at all.

The automatic startup process of VampirServer is performed by a script agent located at *<install-dir>/etc/proxy/agent*. Use the *--server-script=filepath* option to specify an alternative script agent. Make sure that the alternative file path points to a valid script location on the remote system.

```
$ vampir-proxy --server-script=/home/fred/proxy-agent
↪ bedrock.eu
```

A. Appendix

A.1. Default MPI Launch Script

```
#
# Copyright (c) 2011  ZIH, Technische Universitaet Dresden, Germany
#
# @file launcher/mpi
#
# @brief Pure MPI startup sequence without batch job creation.
#
# @author Holger Brunst
#


# System's launch program for MPI programs.
MPIRUN="mpirun"


# Startup timeout in seconds.
TIMEOUT=5

# Launch vampir server process.
launch_vs() {
    ${MPIRUN} ${PREFIX}/bin/vampirserver-core ${opt} >${tmpfile} 2>&1 &
    disown

  # Return process ID or job ID as shutdown reference.
    OUT_CUSTOM=$!
}


# Kill vampir server process.
kill_vs() {
  # Kill server process/job.
    kill -9 ${IN_CUSTOM} 2>/dev/null

  # Wait for server to terminate.
    while [ $(kill -0 "${IN_CUSTOM}" 2>/dev/null) ]; do
        sleep 1
    done
}


# Return the path of a global temporary file.
tmpfile()
{
  # IN_SERVER = internal job ID for server identification

    echo "${HOME}/.vampir/tmp/vampirserver.${IN_SERVER}.tmp"
}


# Parse non-Vampir command line arguments for customization.
IN_CUSTOM_ARG_A=""
IN_CUSTOM_ARG_B=""
```

```
53  parse_custom_args()
54  {
55      temp=`getopt a: "$@"`
56      if [ $? != 0 ] ; then echo "Terminating..." >&2 ; exit 1 ; fi
57      eval set -- "$temp"
58      while true ; do
59          case "$1" in
60              -a) IN_CUSTOM_ARG_A="${2}"; shift 2
61                  ;;
62              --) shift; break
63                  ;;
64              *)  echo "Error: Internal!" >&2; exit 1
65                  ;;
66          esac
67      done
68      if [ -n "$1" ]; then
69          IN_CUSTOM_ARG_B="$1"
70      fi
71  }
72
73
74  # Terminate the given server.
75  stop()
76  {
77    # IN_HOST = IP address of host
78    # IN_PORT = listen port of host
79    # IN_SERVER = internal job ID for server identification
80    # IN_CUSTOM = Launcher custom data. Here: PID of mpirun
81
82    # Terminate VampirServer.
83      kill_vs
84
85    # Clean up temporary output file.
86      rm -f "`tmpfile ${IN_SERVER}`"
87  }
88
89
90  # Start a new server instance.
91  start()
92  {
93    # IN_FORWARD = options to be forwarded to server
94    # IN_NTASKS = number of analysis tasks
95    # IN_SERVER = internal job ID for server identification
96    # IN_TIMEOUT = timeout of startup process
97    # IN_CUSTOM = custom arguments from the command line
98    # OUT_MESSAGE = server output
99    # OUT_CUSTOM = launcher custom data
100   # $? = true if successful, false otherwise
101
102     local success=false
103     local tmpfile="`tmpfile ${IN_SERVER}`"
104
105     if [ -z "${VAMPIRSERVER_DRIVER}" ]; then
106         export VAMPIRSERVER_DRIVER="${PREFIX}/lib/vampirserver-driver.so"
107     fi
108     export VAMPIRSERVER_MODE="mpi-mode"
109
110
111   # Use default for startup timeout?
112     if [ "${IN_TIMEOUT}" ]; then
113         TIMEOUT="${IN_TIMEOUT}"
114     fi
115
116
117   # Use default for number of analysis tasks?
118     if [ -z "${IN_NTASKS}" ]; then
```

```
119         IN_NTASKS=4
120       fi
121
122
123     # Detect MPI implementation. Make sure that VAMPIRSERVER_DRIVER and VAMPIRSERVER_MODE
124     # are exported to all MPI ranks.
125       if (${MPIRUN} --version 2>&1 | grep -q "Open MPI"); then
126         # Open MPI
127         MPIRUN="${MPIRUN} -x VAMPIRSERVER_DRIVER -x VAMPIRSERVER_MODE"
128       fi
129
130
131     # Set number of MPI ranks.
132       MPIRUN="${MPIRUN} -np $((${IN_NTASKS} + 1))"
133
134
135     # Forward options.
136       opt="${opt} ${IN_FORWARD}"
137
138
139     # Example custom arguments.
140       parse_custom_args ${IN_CUSTOM}
141       if [ -n "${IN_CUSTOM_ARG_A}" ]; then
142           echo "Custom argument A = ${IN_CUSTOM_ARG_A}"
143       fi
144       if [ -n "${IN_CUSTOM_ARG_B}" ]; then
145           echo "Custom argument B = ${IN_CUSTOM_ARG_B}"
146       fi
147
148
149     # Launch vampir server in background.
150       launch_vs
151
152
153     # Wait and read server output.
154       local begin=`date +%s`
155       while true; do
156           local listen
157           listen=`grep 'Server listens on:' ${tmpfile} 2>/dev/null`
158
159         # Stop this loop when server has printed his communication link.
160           if [ $? == 0 ]; then
161               success=true
162               break
163           fi
164
165         # Stop this loop when server exited with an error code.
166           if grep -q 'Error: ' ${tmpfile} 2>/dev/null; then
167               break
168           fi
169
170         # Check timeout.
171           if timeout ${begin} ${TIMEOUT}; then
172               break
173           fi
174       done
175
176
177     # Buffer server output.
178       OUT_MESSAGE=""
179       if [ -e "${tmpfile}" ]; then
180           OUT_MESSAGE="`cat ${tmpfile}`"
181       fi
182
183
184     # Kill server in case of a failure.
```

```
185     if ! ${success}; then
186         IN_CUSTOM=${OUT_CUSTOM}
187         stop
188     fi
189
190
191     ${success}
192 }
```

A.2. MPI Launch Script with LSF Support

```
1  #
2  # Copyright (c) 2011  ZIH, Technische Universitaet Dresden, Germany
3  #
4  # @file launcher/lsf
5  #
6  # @brief MPI startup sequence with LSF job creation.
7  #
8  # @author Holger Brunst
9  #
10
11
12 # System's launch program for MPI programs.
13 MPIRUN="mpirun"
14
15 # Custom batch job arguments
16 BATCH_OPT=""
17
18 # Startup timeout in seconds.
19 TIMEOUT=300
20
21 # Launch vampir server process.
22 launch_vs() {
23     local submission; echo "Submitting LSF batch job (this might take a while)..."
24     submission=`bsub -n $((${IN_NTASKS} + 1)) -o ${tmpfile} ${BATCH_OPT} ${MPIRUN} ${
           PREFIX}/bin/vampirserver-core ${opt}`
25
26   # Return process ID or job ID as shutdown reference.
27     OUT_CUSTOM=`echo "${submission}" | grep "is submitted" | sed "s/^Job <//;s/> is
           submitted.*//"`
28 }
29
30
31 # Kill vampir server process.
32 kill_vs() {
33   # Kill server process/job.
34     bkill >/dev/null ${IN_CUSTOM}
35
36   # Wait for server to terminate.
37     while false; do # No waiting needed! Already done by bkill.
38         sleep 1
39     done
40 }
41
42
43 # Return the path of a global temporary file.
44 tmpfile()
45 {
46   # IN_SERVER = internal job ID for server identification
47
48     echo "${HOME}/.vampir/tmp/vampirserver.${IN_SERVER}.tmp"
49 }
50
51
```

```
52  # Parse non-Vampir command line arguments for customization.
53  IN_CUSTOM_ARG_A=""
54  IN_CUSTOM_ARG_B=""
55  parse_custom_args()
56  {
57      temp=`getopt a: "$@"`
58      if [ $? != 0 ] ; then echo "Terminating..." >&2 ; exit 1 ; fi
59      eval set -- "$temp"
60      while true ; do
61          case "$1" in
62              -a) IN_CUSTOM_ARG_A="${2}"; shift 2
63                  ;;
64              --) shift; break
65                  ;;
66              *)  echo "Error: Internal!" >&2; exit 1
67                  ;;
68          esac
69      done
70      if [ -n "$1" ]; then
71          IN_CUSTOM_ARG_B="$1"
72      fi
73  }
74
75
76  # Terminate the given server.
77  stop()
78  {
79    # IN_HOST = IP address of host
80    # IN_PORT = listen port of host
81    # IN_SERVER = internal job ID for server identification
82    # IN_CUSTOM = Launcher custom data. Here: PID of mpirun
83
84    # Terminate VampirServer.
85      kill_vs
86
87    # Clean up temporary output file.
88      rm -f "`tmpfile ${IN_SERVER}`"
89  }
90
91
92  # Start a new server instance.
93  start()
94  {
95    # IN_FORWARD = options to be forwarded to server
96    # IN_NTASKS = number of analysis tasks
97    # IN_SERVER = internal job ID for server identification
98    # IN_TIMEOUT = timeout of startup process
99    # IN_CUSTOM = custom arguments from the command line
100   # OUT_MESSAGE = server output
101   # OUT_CUSTOM = launcher custom data
102   # $? = true if successful, false otherwise
103
104     local success=false
105     local tmpfile="`tmpfile ${IN_SERVER}`"
106
107     if [ -z "${VAMPIRSERVER_DRIVER}" ]; then
108         export VAMPIRSERVER_DRIVER="${PREFIX}/lib/vampirserver-driver.so"
109     fi
110     export VAMPIRSERVER_MODE="mpi-mode"
111
112
113   # Use default for startup timeout?
114     if [ "${IN_TIMEOUT}" ]; then
115         TIMEOUT="${IN_TIMEOUT}"
116     fi
117
```

```
118
119   # Use default for number of analysis tasks?
120     if [ -z "${IN_NTASKS}" ]; then
121         IN_NTASKS=4
122     fi
123
124
125   # Detect MPI implementation. Make sure that VAMPIRSERVER_DRIVER and VAMPIRSERVER_MODE
126   # are exported to all MPI ranks.
127     if (${MPIRUN} --version 2>&1 | grep -q "Open MPI"); then
128       # Open MPI
129         MPIRUN="${MPIRUN} -x VAMPIRSERVER_DRIVER -x VAMPIRSERVER_MODE"
130     fi
131
132
133   # Set number of MPI ranks.
134     MPIRUN="${MPIRUN} -np $((${IN_NTASKS} + 1))"
135
136
137   # Forward options.
138     opt="${opt} ${IN_FORWARD}"
139
140
141   # Example custom arguments.
142     parse_custom_args ${IN_CUSTOM}
143     if [ -n "${IN_CUSTOM_ARG_A}" ]; then
144         echo "Custom argument A = ${IN_CUSTOM_ARG_A}"
145     fi
146     if [ -n "${IN_CUSTOM_ARG_B}" ]; then
147         echo "Custom argument B = ${IN_CUSTOM_ARG_B}"
148     fi
149
150
151   # Launch vampir server in background.
152     launch_vs
153
154
155   # Wait and read server output.
156     local begin=`date +%s`
157     while true; do
158         local listen
159         listen=`grep 'Server listens on:' ${tmpfile} 2>/dev/null`
160
161       # Stop this loop when server has printed his communication link.
162         if [ $? == 0 ]; then
163             success=true
164             break
165         fi
166
167       # Stop this loop when server exited with an error code.
168         if grep -q 'Your job looked like:\|Error:' ${tmpfile} 2>/dev/null; then
169             break
170         fi
171
172       # Check timeout.
173         if timeout ${begin} ${TIMEOUT}; then
174             break
175         fi
176     done
177
178
179   # Buffer server output.
180     OUT_MESSAGE=""
181     if [ -e "${tmpfile}" ]; then
182         OUT_MESSAGE="`cat ${tmpfile}`"
183     fi
```

```
184
185
186     # Kill server in case of a failure.
187       if ! ${success}; then
188           IN_CUSTOM=${OUT_CUSTOM}
189           stop
190       fi
191
192
193       ${success}
194   }
```